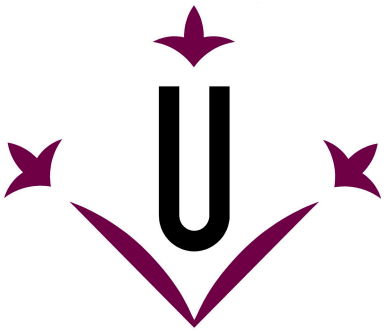


Implementació d'un protocol segur per a jocs distribuïts amb llançament de daus

Autor: Xavier Moya Farré

Director: Francesc Sebé Feixas

Universitat de Lleida

Escola Politècnica Superior

Grau en Enginyeria Informàtica

Treball Final de Grau

Setembre de 2017

Index

Index de figures

Introducció

Aquest treball consta de dues parts. En primer lloc s'ha desenvolupat un protocol de llançament de dau segur per jocs distribuïts. És un protocol el qual permet que dos o més jugadors separats geogràficament puguin dur a terme tirades de dau totalment atzaroses i sense cap possibilitat de que ningú esbiaixi ni pugui preveure el resultat. Això s'ha fet amb un protocol que utilitza criptografia simètrica. La segona part del treball consisteix en desenvolupar una aplicació per a dispositius mòbils sobre la plataforma Android. Aquest joc simula tirades de daus entre els jugadors seguint el protocol implementat en la primer part. A més a més es fa ús de "Google Play Games Services" per tal de crear partides i manejar els participants.

La gran part del treball és pràctica. Juntament amb aquest document es proporciona l'aplicació Android que és podrà descarregar al smartphone i també s'entrega una còpia del codi. Per aquesta raó, part del document mostra el codi desenvolupat; es mostren algunes línies de codi i s'expliquen.

Algorisme de llançament de dau segur

La finalitat de l'algorisme és que en un joc multi-jugador en que cada jugador ha de tirar un dau del qual depèn l'èxit en el joc, no es pugui, de cap manera, esbiaixar el resultat del dau per afavorir-se o perjudicar al contrincant. És a dir, volem aconseguir un llançament del dau cent per cent segur.

Per aquest motiu s'ha implementat un algorisme que ens garanteix la fiabilitat de les tirades.

Anem a veure-ho.

Suposant el cas que tenim una partida de dos jugadors i és el torn del jugador 1.

- El jugador 1 genera a l'atzar un valor entre 0 i 5.
- El jugador 1 xifra el resultat.
- El jugador 1 envia al jugador 2 el missatge xifrat.
- El jugador 2 rep el missatge xifrat del jugador 1.
- El jugador 2 també genera un valor entre 0 i 5.
- El jugador 2 xifra el resultat.
- El jugador 2 envia al jugador 1 el missatge xifrat.
- El jugador 1 rep el missatge xifrat del jugador 2.
- El jugador 1 envia al jugador 2 la clau per desxifrat el seu missatge.
- El jugador 2 envia al jugador 1 la clau per desxifrar el seu missatge.
- El jugador 1 desxifra el missatge del jugador 2.
- El jugador 2 desxifra el missatge del jugador 1.
- El jugador 1 té dos valors, així que els utilitzará per esbrinar quin és realment el valor de la seva tirada.
- El jugador 2 té dos valors, així que els utilitzará per esbrinar quin és realment el valor de la tirada del jugador 1.
- Un cop els dos jugadors acorden quin ha estat el valor del dau pot continuar la partida.

Ara és el torn del jugador 2. Es farà el mateix que s'ha fet amb el torn del jugador 1.

La pregunta ara seria, com s'esbrina el valor del dau si tenim dos valors? (o més de dos, tants com jugadors):

- Primer es sumen tots els valors.
- Al valor resultant se l'hi fa un modul 6 (nombre de possibles valors que té el dau)
- El resultat és el valor del dau.

Vegem per què hem fet un algorisme tan enrevessat i per què podem dir que es totalment segur i no es pot esbiaixar el resultat.

Per evitar que algú interfereixi el missatge i canviï el resultat el què fem és xifrar el missatge. A més evitem la inconsistència de les dades, ja que arriba un resultat diferent de l'enviat.

En primera instància algú podria pensar que pot fer que el valor del dau sigui un en concret només enviant aquest valor, per aquesta raó fem que el resultat de la tirada depengui de tots els jugadors enlloc de només del jugador que té el torn. Si no fos així ell podria dir que ha tret el valor que volgués només enviant als altres aquest valor.

Si utilitzéssim un altre algorisme, es podria intentar forçar un valor petit o un valor elevat. Per exemple si el nostre algorisme fes una mitja dels valors només ens caldria enviar un 0 per forçar que el resultat final sigui baix. Però el nostre algorisme fa un mòdul de la suma. Així aconseguim que independentment del valor que vulguem forçar, serà impossible i aconseguim sempre una tirada totalment atzarosa i segura.

Vegem el procés d'un torn amb imatges:

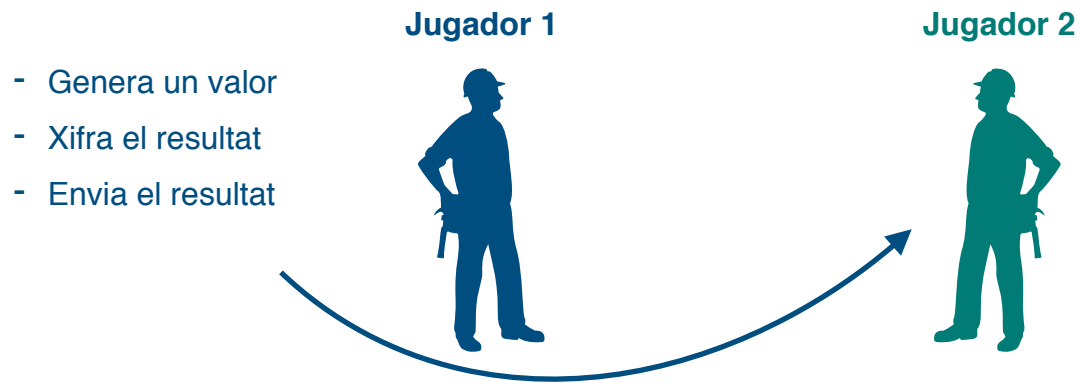


FIGURA 1: TORN DE LA PARTIDA - 1

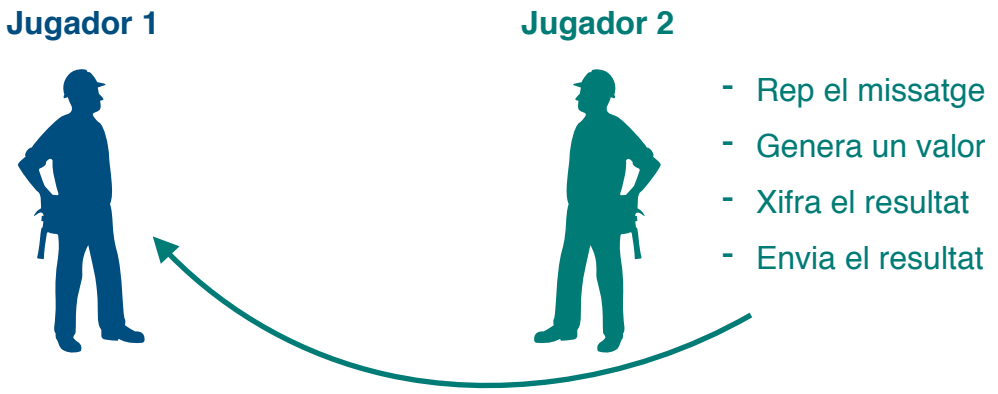


FIGURA 2: TORN DE LA PARTIDA - 2

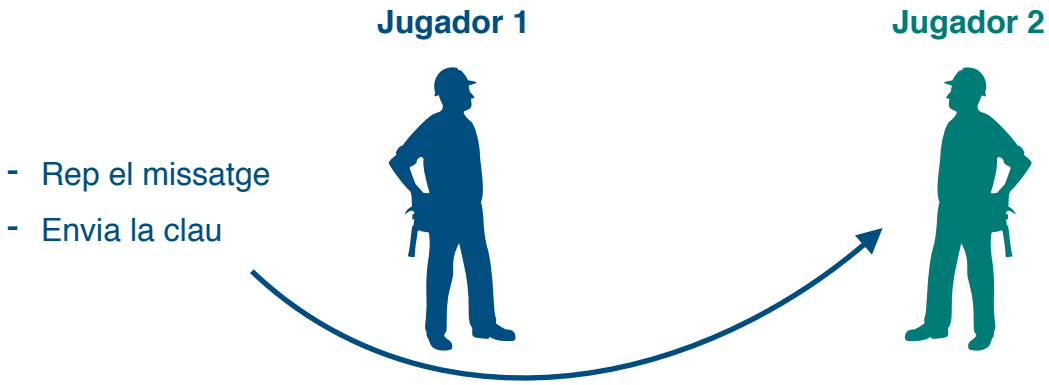


FIGURA 3: TORN DE LA PARTIDA - 3

FIGURA 4: TORN DE LA PARTIDA - 4

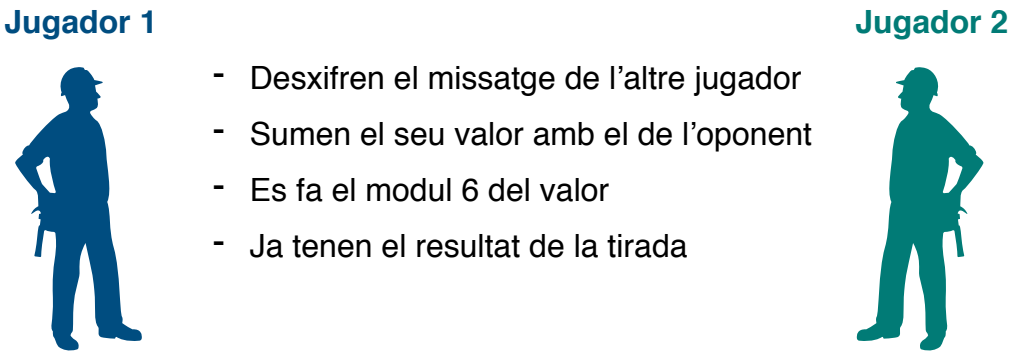


FIGURA 5: TORN DE LA PARTIDA - 5

Ara seria el torn del jugador 2 i es faria el mateix.

Aquest algorisme es pot generalitzar a qualsevol nombre de jugadors. Per generalitzar-lo haurem de fer els enviaments a tots els jugadors enlloc de només al "Jugador 2". A més, quan rebem un missatge, el rebrem també de tots ells. Haurem d'esperar que cada pas el facin tots els jugadors, enlloc de sols un. Tindrem més valors i més claus, tants com nombres de jugadors, i a l'hora de fer el càlcul del valor final farem ús de tots els valors.

Implementació de l'algorisme

La implementació l'he fet per a la plataforma Android, així què el llenguatge de programació escollit és Java.

He implementat uns quants mètodes necessaris per l'algorisme a la classe principal y després he creat una classe auxiliar anomenada "SimpleCrypto.java" on hi ha altres funcions addicionals utilitzades per la criptografia.

Classe principal

A continuació descrivim els mètodes de la classe principal importants per l'algorisme:

firstStep()

És el primer pas de l'algorisme. Es comprova si el dau s'ha tirat ja, i si no és així es calcula el valor de la tirada, es xifra i s'envia.

```
void firstStep() throws Exception {
      if(!firstSended){
          firstSended = true;
          diceValue = diceResult();
         String seed = randomSeed();
         key = SimpleCrypto.getRawKey(seed.getBytes());
         String cifrado = SimpleCrypto.encrypt(key, String.valueOf(diceValue));
        sendDiceValue(cifrado);
      }
}
```

La variable "firstSended" ens serveix com una espècie de semàfor per saber si ja hem fet la tirada o no.

El mètode "randomSeed()" genera una cadena alfanumèrica aleatòria per utilitzar-la com a llavor per al xifrage.

El mètode "sendDiceValue(cifrado)" s'encarrega d'enviar el valor "cifrado" que és el valor del dau xifrat. També tenim un mètode per enviar la clau.

diceResult()

Simula el llançament d'un dau.

```
private int diceResult() {
      Random r = new Random();
      return r.nextInt(MAX_VALUE - MIN_VALUE + 1) + MIN_VALUE;
}
```

Els valors “MAX_VALUE” i “MIN_VALUE” estableixen els valors màxims i mínims del dau, es a dir 0 i 5 (que representen 1 i 6 en un dau real).

receiveDice(byte[] buf)

Es guarda el valor rebut del dau, el qual està xifrat.

```
void receiveDice(byte[] buf){
     //Save the encypted dice value of the opponent
     byte[] cypherdice = new byte[buf.length-1];
     int counter = 1;
     while (counter < buf.length){
          cypherdice[counter-1] = buf[counter];
          counter++;
     }
     opponentDice = SimpleCrypto.fromHex(SimpleCrypto.toHex(cypherdice));
}
```

receiveKey(byte[] buf)

Es guarda el valor rebut de la clau i ara que ja ho tenim tot, desxifrem el valor xifrat del dau que hem guardat abans. Finalment es calcula el valor final.

```
void receiveKey(byte[] buf){
     //Now, I receive key and I can decode the value
     byte[] opponentKey = new byte[buf.length-1];
      int counter = 1;
      while (counter < buf.length){
           opponentKey[counter-1] = buf[counter];
           counter++;
      }

      //With key and dice I can calculate the result dice
      try {
         String valueOpponent = SimpleCrypto.decrypt(opponentKey, opponentDice);
         calculateFinalValue(diceValue+1,Integer.parseInt(valueOpponent)+1);
     } catch (Exception e) {
         e.printStackTrace();
     }
}
```

calculateFinalValue(int v1, int v2)

S’encarrega de calcular el valor final del dau un cop tenim els valors de dau dels dos jugadors desxifrats. És la part final de l’algorisme.

```
void calculateFinalValue(int v1, int v2){
       int val = v1+v2;
       val = val % (MAX_VALUE + 1);  //+1 to real max value of dice =6

       setImageValue(val);
}
```

El mètode “setImageValue(val)” s’encarrega de canviar l’imatge del dau en l’aplicació Android per mostrar el valor obtingut.

Classe “SimpleCrypto.java”

Necessitem importar llibreries de suport per la criptografia.

```
import java.ecurity.SecureRandom;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
```

Vector de inicialització

En primer lloc tenim un vector de inicialització per tal de poder dur a terme el xifratge i desxifratge dels texts.

```
private static String iv = "0123456789ABCDEF";
```

encrypt(byte[] rawKey, String cleartext)

És una funció pública emprada per xifrar un text en clar “cleartext” mitjançant una clau “rawKey”.

```
public static String encrypt(byte[] rawKey, String cleartext) throws Exception {
      byte[] result = encrypt(rawKey, cleartext.getBytes());
      return toHex(result);
}
```

decrypt(byte[] rawKey, String encrypted)

Aquesta funció pública la utilitzem per desxifrar un text xifrat “encrypted” mitjançant la clau “rawKey”.

```
public static String decrypt(byte[] rawKey, String encrypted) throws Exception {
      byte[] enc = toByte(encrypted);
      byte[] result = decrypt(rawKey, enc);
```

```
        return new String(result);
}
```

getRawKey(byte[] seed)

Aquest mètode s'utilitza per generar una clau aleatòria per poder utilitzar-la per xifrar.

Aquesta clau la creem de 256 bits per xifrar amb l'algorisme de xifratge AES (Advanced Encryption Standard) i el hash SHA1.

```
public static byte[] getRawKey(byte[] seed) throws Exception {
        KeyGenerator kgen = KeyGenerator.getInstance("AES");
        SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
        sr.setSeed(seed);
        kgen.init(256, sr); // 192 and 256 bits may not be available
        SecretKey skey = kgen.generateKey();
        return skey.getEncoded();
}
```

encrypt(byte[] raw, String clear)

És una funció emprada per xifrar un text en clar "cleartext" mitjançant una clau "raw" i el vector de inicialització.

Per xifrar s'utilitza l'algorisme estàndard de xifratge AES (Advanced Encryption Standard) amb el mode d'operació CBC (Cipher Block Chaining).

```
private static byte[] encrypt(byte[] raw, byte[] clear) throws Exception {
        SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        IvParameterSpec ivParameterSpec = new IvParameterSpec(iv.getBytes());
        cipher.init(Cipher.ENCRYPT_MODE, skeySpec, ivParameterSpec);
        return cipher.doFinal(clear);
}
```

decrypt(byte[] raw, String encrypted)

Aquesta funció la utilitzem per desxifrar un text xifrat "encrypted" mitjançant la clau "rawKey" i el vector de inicialització.

Per desxifrar s'utilitza el algorisme estàndard de xifratge AES (Advanced Encryption Standard) amb el mode d'operació CBC (Cipher Block Chaining).

```
private static byte[] decrypt(byte[] raw, byte[] encrypted) throws Exception {
        SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        IvParameterSpec ivParameterSpec = new IvParameterSpec(iv.getBytes());
        cipher.init(Cipher.DECRYPT_MODE, skeySpec, ivParameterSpec);
```

```
        return cipher.doFinal(encrypted);
}
```

toHex(String txt)

Aquesta funció s'utilitza per passar un text a hexadecimal.

```
public static String toHex(String txt) {
      return toHex(txt.getBytes());
}
```

fromHex(String hex)

Aquesta funció s'utilitza per passar un text hexadecimal a text ASCII normal.

```
public static String fromHex(String hex) {
      return new String(toByte(hex));
}
```

toByte(String hexString)

Aquesta funció s'utilitza per passar un text en hexadecimal a bytes.

```
private static byte[] toByte(String hexString) {
        int len = hexString.length() / 2;
        byte[] result = new byte[len];
        for (int i = 0; i < len; i++)
             result[i] = Integer.valueOf(hexString.substring(2 * i, 2 * i + 2),
             16).byteValue();
        return result;
}
```

toHex(byte[] buf)

Aquesta funció s'utilitza per passar un text en bytes a hexadecimal.

```
private static String toHex(byte[] buf) {
        if (buf == null)
             return "";
        StringBuffer result = new StringBuffer(2 * buf.length);
        for (byte aBuf : buf) {
             appendHex(result, aBuf);
        }
        return result.toString();
}
```

Cadena de caràcters hexadecimals

Aquesta cadena la utilitzarem en el pròxim mètode, en el que necessitem els valors hexadecimals.

```
private final static String HEX = "0123456789ABCDEF";
```

appendHex(StringBuffer sb, byte b)

Aquest mètode s'utilitza per convertir els bytes a valors hexadecimal. El fem servir en la funció "toHex(byte[] buf)" anterior i necessitem la cadena "HEX" per tenir els valors hexadecimals per tal de fer la conversió.

```
private static void appendHex(StringBuffer sb, byte b) {
	sb.append(HEX.charAt((b >> 4) & 0x0f)).append(HEX.charAt(b & 0x0f));
}
```

Criptografia emprada

La paraula criptografia (o criptologia) es una paraula composta per dos paraules que provenen del grec, Kryptos, que vol dir “amagat, secret” i gràphin, que vol dir “escriptura” (o -λογία, -logia que significa “estudi”).

La criptografia és l’estudi de com convertir informació en un codi impossible d’entendre. Així aconseguim que sigui incomprensible per tothom que no conegui la tècnica emprada. Actualment s’utilitzen les disciplines de les matemàtiques, l’electrotècnia i l’informàtica.

Xifratge

El xifratge, també anomenat encriptació, és el procediment amb el que s’amaga la informació emprant un codi secret o un mètode de xifratge. Així s’aconsegueix que el missatge inicial sigui difícil o gairebé impossible de reconèixer, excepte si es té la clau secreta per poder desxifrar-lo.

Principalment tenim 2 tipus de xifratge:

- **Xifratge simètric:** Direm que es xifratge simètric quan utilitzem la mateixa clau per xifrar i desxifrar.
- **Xifratge asimètric:** Direm que es xifratge asimètric quan utilitzem una parella composta de dues claus, una clau pública, per xifrar, i una clau privada per desxifrar.

Ens centrarem en l’algorisme emprat en la pràctica, aquest s’anomena AES i és un tipus de xifratge simètric.

Algorisme AES

Aquest mètode de xifratge s'anomena AES (de l'anglès "Advanced Encryption Standard”). També es coneix com Rijndael (que es pronuncia "Rain Doll", també en anglès), que tot i no ser exactament el mateix, és d'on ha sortit AES amb una sèrie de modificacions.. Es tracta d'un xifratge per blocs que es va adoptar com a estàndard pels Estats Units el 26 de maig de 2002, tot i que va ser presentat el 26 de novembre del 2001. El seu procés d’estandardització va trigar 5 anys.

Esquema de les etapes

Per poder xifrar amb l’algorisme AES tenim que seguir uns passos concrets, els quals anomenem etapes. Les podem veure a continuació:

• Expansió de la clau utilitzant l'esquema de claus de Rijndael.
• Etapa inicial:
    1. AddRoundKey
• Rondes:
    1. SubBytes - en aquest pas es realitza una substitució no lineal on cada byte és reemplaçat per un altre d'acord amb una taula de cerca.
    2. ShiftRows - en aquest pas es realitza una transposició on cada fila del "state» és Rotat de manera cíclica un nombre determinat de vegades.
    3. MixColumns - operació de barreja que opera a les columnes del «state», combinant els quatre bytes de cada columna utilitzant una transformació lineal.
    4. AddRoundKey - cada byte de l'«state» és combinat amb la clau «round»; cada clau «round» es deriva de la clau de xifratge utilitzant una iteració de la clau.
• Etapa final:
    1. SubBytes
    2. ShiftRows
    3. AddRoundKey

Podem veure el procés una mica més clar a la figura següent:

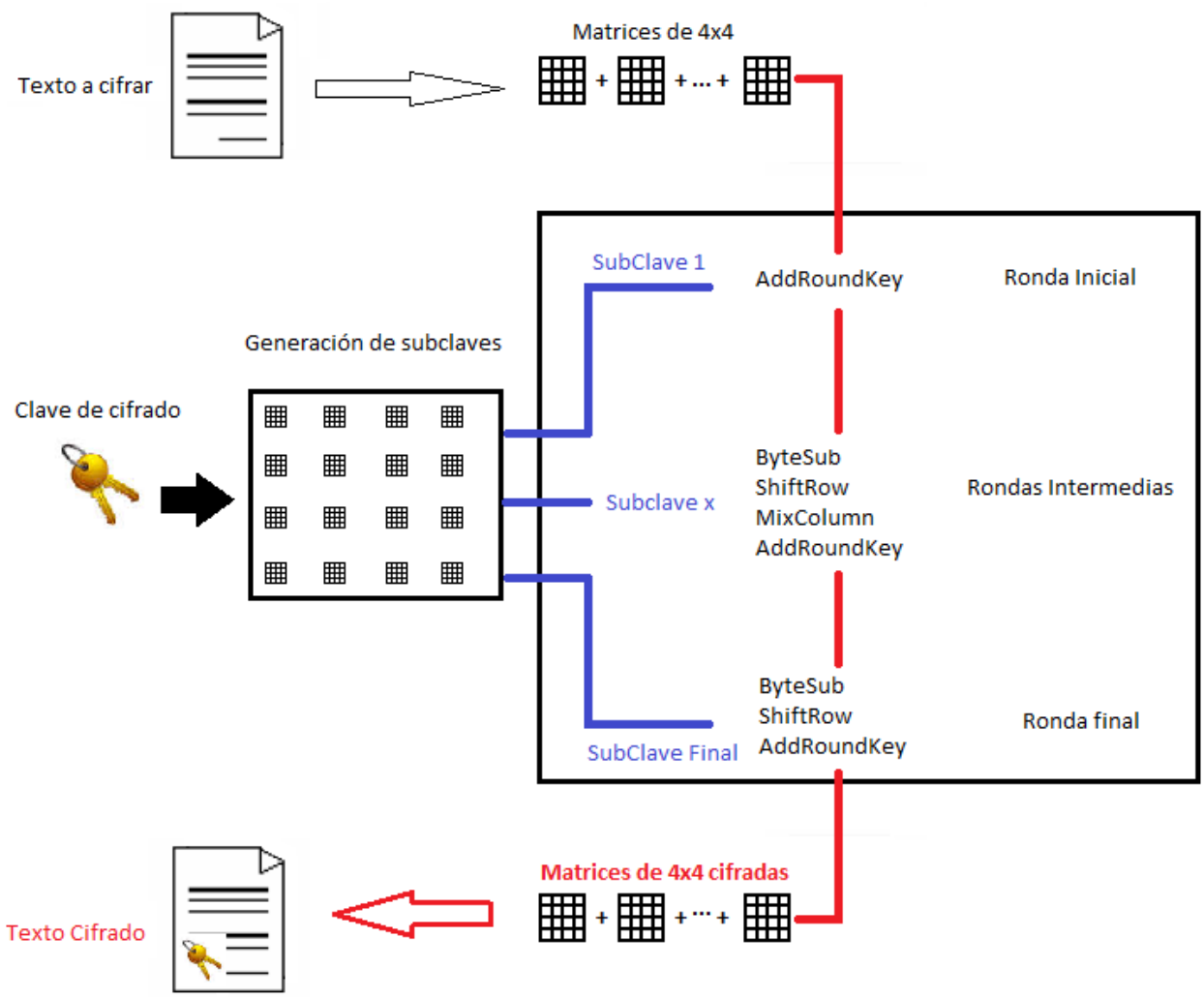
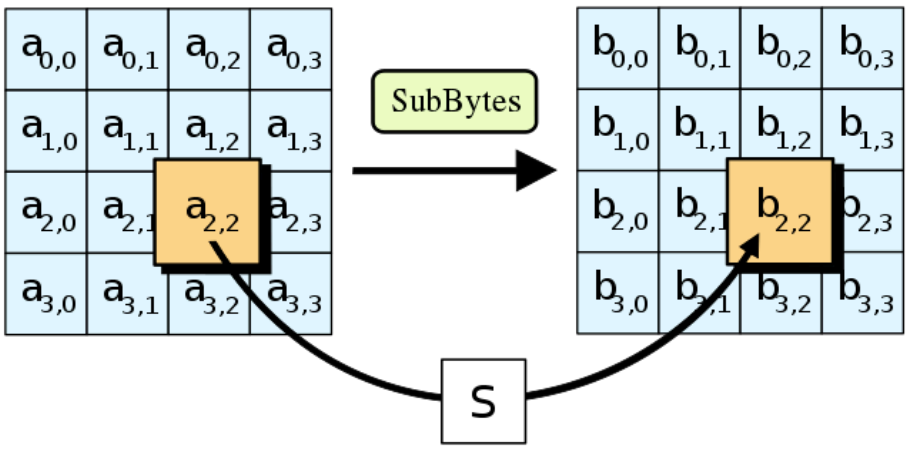


FIGURA 6: ESQUEMA DE XIFRATGE AES. RECUPERAT DE HTTP://WWW.DMDCOSILLAS.ORG/2014/08/CIFRADO-DE-PARTICIONES-ALGORITMO-AES-I/

A continuació veurem cada etapa per separat.

Etapa SubBytes - Substitució de bits

En l'etapa SubBytes , cada byte de matriu és actualitzat utilitzant la caixa-S de Rijndael de 8 bits. Gràcies a aquesta operació aconseguim la no linealitat en el xifratge. La caixa-S utilitzada prové de la funció inversa al voltant del GF (2^8), conegut per tenir grans propietats de no linealitat. Per evitar atacs basats en simples propietats algebraiques, la caixa-S es construeix per la combinació de la funció inversa amb una transformació afí invertible. La caixa-S també es tria per evitar punts estables, i també els punts estables oposats.



En la fase de SubBytes , cada byte al state és reemplaçat amb la seva entrada en una taula de cerca fixa de 8 bits, S ; b ij = S (a ij) .

FIGURA 7: ETAPA SUBBYTES. RECUPERAT DE HTTPS://ES.WIKIPEDIA.ORG/WIKI/ADVANCED_ENCRYPTION_STANDARD

Etapa ShiftRows - Desplaçar files

El pas ShiftRows opera a les files del state. Els bytes es trenquen de manera cíclica en cada fila per un determinat offset. En AES, la primera fila queda a la mateixa posició. Cada byte de la segona fila és mogut una posició a l'esquerra. De manera similar, la tercera i quarta fila són rotades dues i tres posicions a l'esquerra respectivament. D'aquesta manera, cada columna de l'state resultant del pas ShiftRows està composada per bytes de cada columna de l'state inicial.

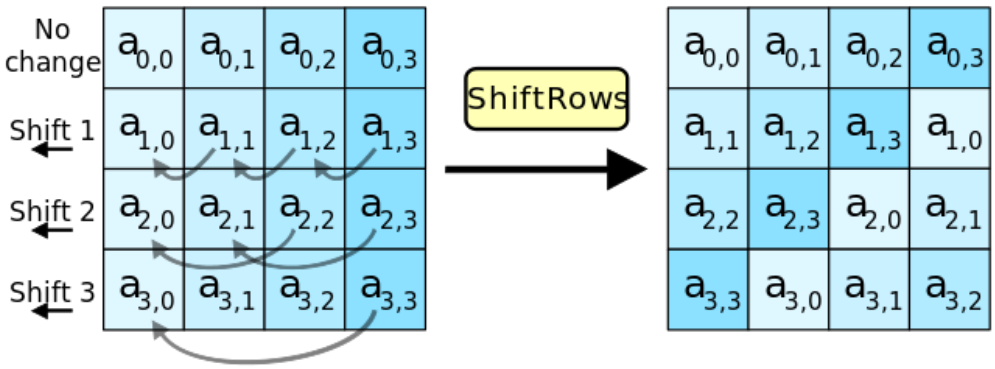
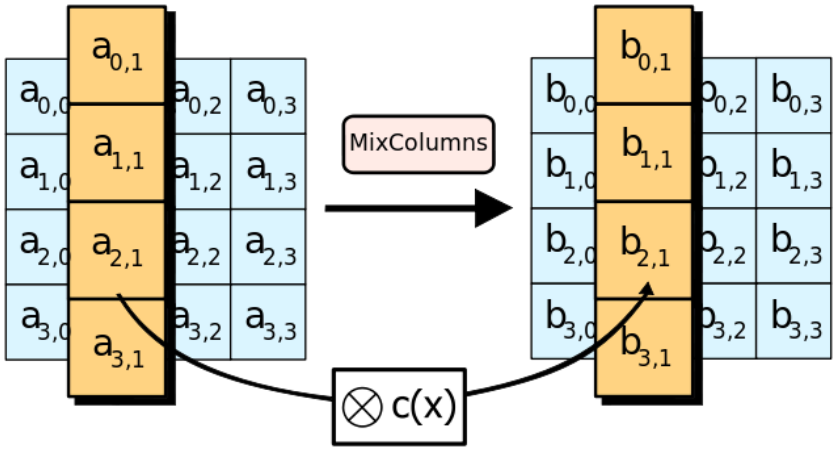


FIGURA 8: ETAPA SHIFTROWS. RECUPERAT DE HTTPS://ES.WIKIPEDIA.ORG/WIKI/ADVANCED_ENCRYPTION_STANDARD

En el pas ShiftRows , els bytes en cada fila de l'state són rotats de manera cíclica cap a l'esquerra. El nombre de llocs que cada byte és rotat difereix per a cada fila.

Etapa MixColumns - Barrejar columnes

En el pas MixColumns , els quatre bytes de cada columna de l'state es combinen utilitzant una transformació lineal invertible. La funció MixColumns pren quatre bytes com a entrada i retorna quatre bytes, on cada byte d'entrada influeix totes les sortides de quatre bytes. Juntament amb ShiftRows , MixColumns implica difusió al xifrat. Cada columna es tracta com un polinomi GF (2^8) i després es multiplica el mòdul amb un polinomi fix . El pas MixColumns es pot veure com una multiplicació matricial al camp finit de Rijndael.

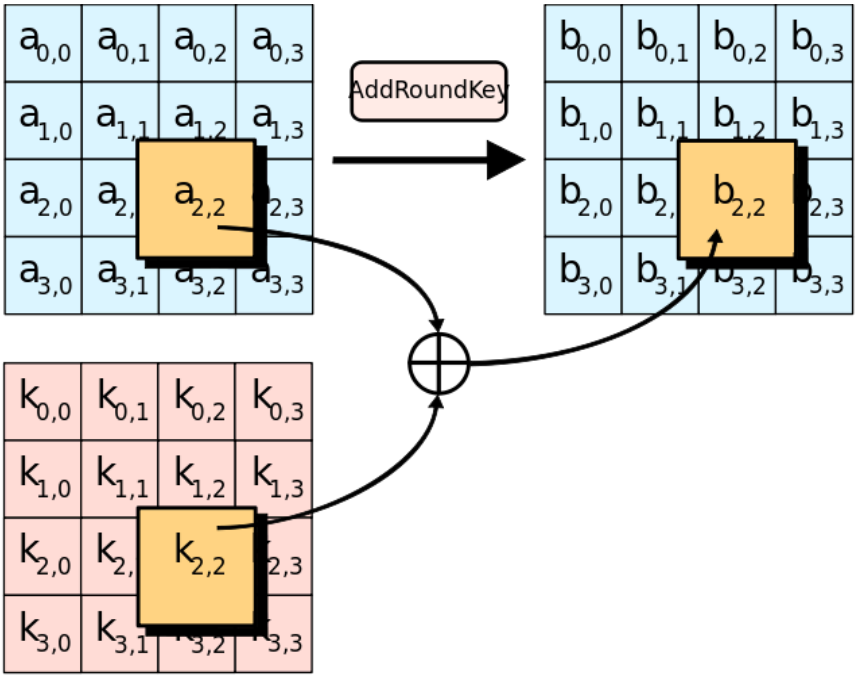


En el pas MixColumns , cada columna de l'state és multiplicada per un polinomi constant c (x) .

FIGURA 9: ETAPA ETAPA MIXCOLUMNS. RECUPERAT DE HTTPS://ES.WIKIPEDIA.ORG/WIKI/ADVANCED_ENCRYPTION_STANDARD

Etapa AddRoundKey - Càlcul de les subclaus

En el pas AddRoundKey , la subclau es combina amb el state. En cada ronda s'obté una subclau de la clau principal, utilitzant la iteració de la clau, cada subclau és de la mateixa mida del state. La subclau s'agrega combinant cada byte de l'state amb el corresponent byte de la subclau utilitzant XOR.



En el pas AddRoundKey , cada byte de l'state es combina amb un byte de la subclau utilitzant l'operació XOR (⊕).

FIGURA 10: ETAPA ADDROUNDKEY. RECUPERAT DE HTTPS://ES.WIKIPEDIA.ORG/WIKI/ADVANCED_ENCRYPTION_STANDARD

Aplicació Android

S'ha desenvolupat l'aplicació en la plataforma Android ja que és de codi obert, és la que hem estudiat en la carrera i és la més popular actualment.

Aquesta aplicació és un joc multi-jugador en temps real. La seva funció consisteix en realitzar una tirada d'un dau simultània entre dos jugadors i que el resultat sigui cent per cent aleatori. És a dir, que no és pugui forçar cap valor.

Ja hem parlat de l'algorisme de llançament de dau segur i també de la criptografia que utilitzem en l'algorisme. Més endavant parlarem d'una part molt important, els serveis de Google que hem utilitzat per controlar les partides. Aquest punt ha estat el que més feina m'ha portat i ho deixarem pel final.

Aquí explicaré en que consisteix l'aplicació, les seves funcions. Ho mostraré amb una sèrie de captures de pantalla.

Parts bàsiques del joc

El jugador, un cop s'ha instal·lat la aplicació i l'obre ha de registrar-se amb el seu compte de Google. Si no ho fa, només podrà utilitzar el mode "un jugador" en el què, per falta de participants, no s'utilitza l'algorisme de llançament segur.

A continuació anem a veure les diferents opcions i pantalles que tenim dins l'aplicació.

Pantalla inicial

La pantalla inicial és diferent si estem registrats o no.

Si no ho estem ens dóna l'opció de registrar-nos o de jugar sols.

Si ens registrem tenim també l'opció de jugar sols a més de poder iniciar un joc ràpid, invitar amics, veure invitacions o tancar la sessió.

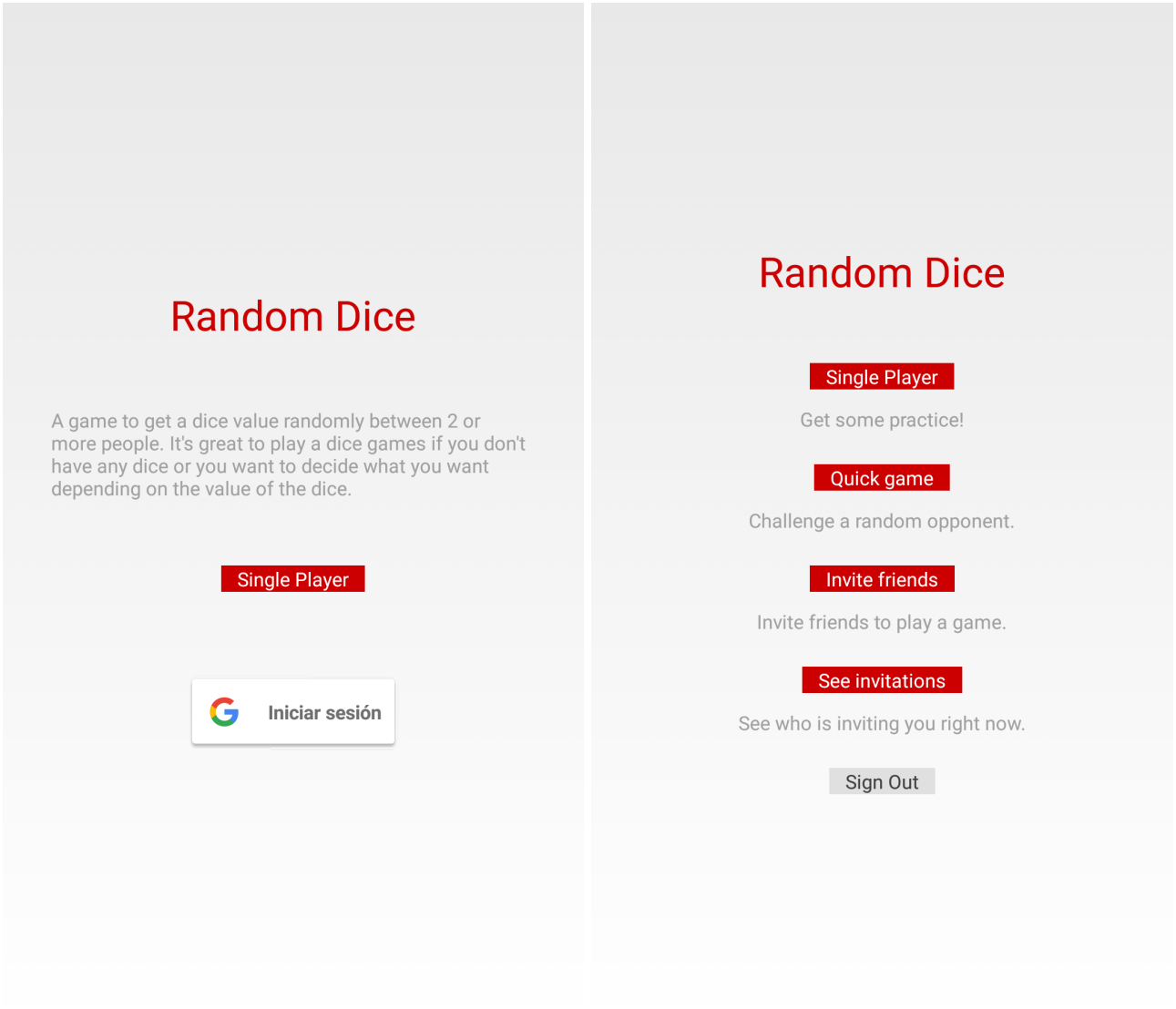


FIGURA 11: SCREENSHOTS PANTALLA PRINCIPAL. SESSIÓ NO INICIADA - SESSIÓ INICIADA, RESPECTIVAMENT

Les instruccions

Les instruccions es mostren a la pantalla inicial quan no estem registrats. Com podem veure a l'anterior figura 11.

El joc serveix per a obtenir un valor d'un dau de manera completament aleatòria entre 2 jugadors. És ideal per jugar a jocs de daus si no disposem d'un dau o si volem decidir alguna cosa de manera aleatòria fent-ho en funció del dau, ja que seguim un algorisme totalment segur que garanteix el factor atzarós.

Invitar

A més podrem invitar a jugar als amics que tinguem afegits al compte de jocs de Google. També podem invitar als últims oponents o buscar algun jugador que no ens aparegui al llistat.

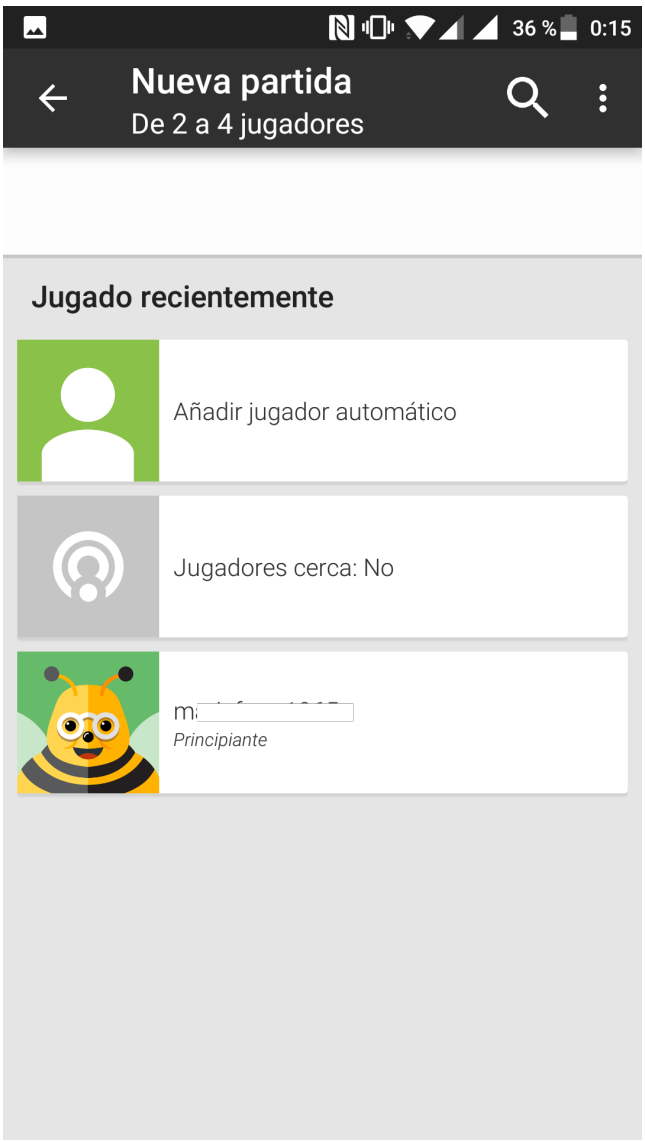


FIGURA 12: SCREENSHOT INVITAR

Veure invitacions

Podem consultar les invitacions que estan vigents per jugar. A més quan algú ens invita es mostra un avís a la part superior de la pantalla.

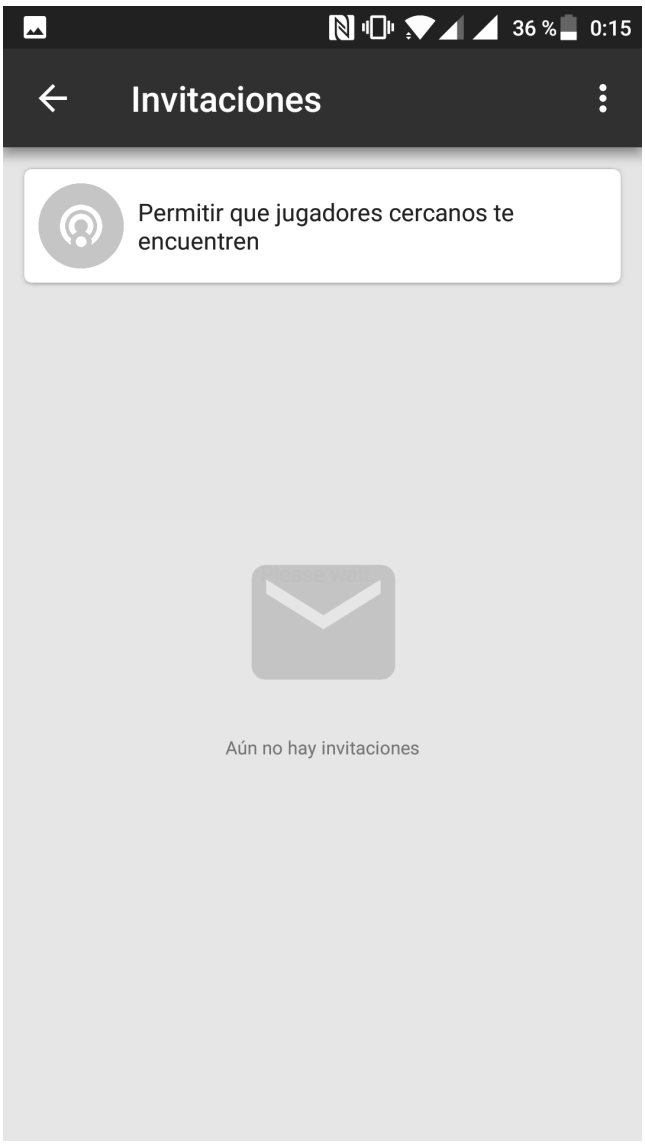


FIGURA 13: SCREENSHOT VEURE INVITACIONS

El joc

Aquesta pantalla ens mostrarà un dau amb el resultat del llançament.

En el moment que veiem aquesta pantalla s'executa en segon pla l'algorisme. Un cop ha acabat ens mostra el valor del dau. El més normal és que directament vegem el resultat ja que és un procès força ràpid i no ens n'adonem.

Llavors només tenim 2 opcions, prémer el botó enrere per tornar a la pantalla inicial o fer clic en "nova tirada" per tornar a tirar el dau. Si és una partida d'un jugador es tira al moment, sinó tornem a fer la cerca automàtica de participants.

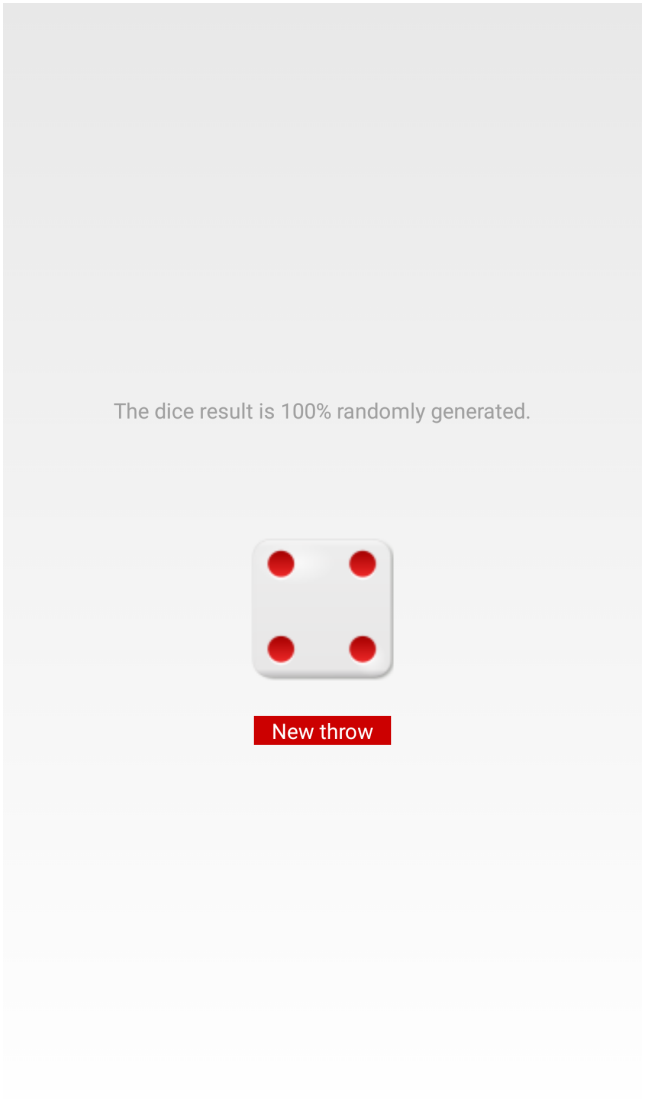


FIGURA 14: SCREENSHOT PANTALLA DEL JOC

Google Play Games Services

Play Games SDK ofereix serveis de jocs multiplataforma de Google Play que ens permet integrar fàcilment les funcions més populars de jocs, com per exemple les taules de classificació, els èxits, jocs guardats i les partides multi-jugador a temps real o basades en torns, tant a jocs per a mòbil com a la tauleta.

En primer lloc, per utilitzar els serveis de Google haurem de fer uns petits preparatius de l'aplicació.

Preparatius aplicació

Abans de començar necessitem l'entorn de treball per desenvolupar una aplicació Android i un dispositiu Android amb un SDK mínim de Android 2.3 (Gingerbread) per a fer el testeig.

Seguint les instruccions de Google el primer pas és descarregar-nos una aplicació exemple que ja té una base de la integració amb els seus serveis feta. En aquesta aplicació haurem d'editar el fitxer "AndroidManifest.xml" i canviar el nom del paquet pel nom de paquet que vulguem escollir per la nostra aplicació.

El segon pas consisteix en donar d'alta i configurar el projecte a la consola de Google Play. La consola de Google Play és el lloc on gestionarem el serveis de jocs de Google Play per al nostre joc i també hi configurarem les metadades per autoritzar i autenticar el nostre joc.

Per dur a terme aquest past haurem de registrar-nos a la consola si no ho hem fet mai. Un cop registrats, hem d'afegir el joc a la consola de Google Play. A continuació podrem generar l'identificador del client "OAuth 2.0" per enllaçar l' aplicació. És molt important fixar-nos que l'aplicació és enllaçada amb el nom del paquet corresponent. Per generar un magatzem de claus i un certificat signat (si no en tenim) podem utilitzar l'assistent de generació d'APK signada que tenim en Android Studio. Haurem de guardar l'identificador d'aplicació i el certificat signat per introduir-lo més endavant.

Ara és el torn de configurar els èxits i les taules de classificació. Per acabar hem d'introduir els comptes de testeig pel joc.

El següent pas consisteix en modificar el codi per incloure la informació que hem introduït a la consola. Al fitxer "res/values/ids.xml" (cal crear-lo si no el tenim) hi posarem els identificadors de l'aplicació, dels èxits i de les classificacions que hem creat. A més, en l' "AndroidManifest.xml" també hi inclourem l'identificador de l'aplicació:

```
<meta-data android:name="com.google.android.gms.games.APP_ID"
     android:value="@string/app_id" />
<meta-data android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version"/>
```

Finalment. l'últim pas és testejar que les modificacions del joc funcionen correctament. Per provar-lo haurem de generar un APK i instal·lar l'aplicació en el dispositiu mòbil Android.

Multi-jugador en temps real

Quan parlem d'un joc multi-jugador en temps real ens referim a què en aquest joc hi ha una comunicació entre els jugadors de manera instantània i que per jugar-hi és imprescindible que tots els participants estiguin connectats alhora. En qualsevol moment un participant pot enviar un missatge a algun dels altres, o a tots.

Google ens dóna suport a un nombre d'entre 2 i 8 jugadors, on s'inclou la cerca automàtica ,que a més, la podem combinar amb els jugadors invitats i així jugar amb qui invitis tu personalment i els jugadors que es trobin de manera aleatòria.

Preparatius multi-jugador en temps real

Un cop hem preparat tot el que s'ha explicat al punt "Preparatius aplicació" encara ens queda configurar un altre petit detall que només serà necessari fer-ho així en cas d'estar configurant un joc en temps real.

A la consola de Google Play haurem d'activar l'interruptor que diu "Multi-jugador a temps real" i desactivar el que diu "Multi-jugador per torns". Si no ho fem, l'aplicació ens donarà errors.

FIGURA 15: INTERRUPTOR MULTI-JUGADOR EN TEMPS REAL

Joc ràpid

Quan el jugador selecciona l'opció de "joc ràpid", es crea una espècie d'habitació virtual on s'uneixen els jugadors, es combina el jugador amb els jugadors cercats automàticament, un cop s'ha ajuntat el grup de jugadors s'inicia el joc immediatament.

```
private void startQuickGame() {
    // auto-match criteria to invite one random automatch opponent.
    // You can also specify more opponents (up to 3).
    Bundle am = RoomConfig.createAutoMatchCriteria(1, 1, 0);

    // build the room config:
    RoomConfig.Builder roomConfigBuilder = makeBasicRoomConfigBuilder();
    roomConfigBuilder.setAutoMatchCriteria(am);
    RoomConfig roomConfig = roomConfigBuilder.build();

    // create room:
    Games.RealTimeMultiplayer.create(mGoogleApiClient, roomConfig);

    // prevent screen from sleeping during handshake
    getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);

    // go to game screen
}
```

Invitar jugadors

Utilitza una pantalla estàndard d'invitacions de Google. Se'ns farà familiar ja que és la que utilitzen la gran majoria dels jocs multi-jugador d'Android.

Fer servir aquesta pantalla d'invitacions és molt fàcil. Només haurem d'utilitzar les línies de codi següents:

```
// request code for the "select players" UI
// can be any number as long as it's unique
final static int RC_SELECT_PLAYERS = 10000;

// launch the player selection screen
// minimum: 1 other player; maximum: 3 other players
Intent intent =
Games.RealTimeMultiplayer.getSelectOpponentsIntent(mGoogleApiClient, 1, 3);
startActivityForResult(intent, RC_SELECT_PLAYERS);
```

És a dir, creem un "intent" i el posem en marxa.

A més ens farà falta un oient per l'altra banda. Així podrem manejar el resultat. Per veure si ha estat cancel·lat:

```
public void onActivityResult(int request, int response, Intent data){
      super.onActivityResult(request, response, data);

      if (response != Activity.RESULT_OK){
            //user canceled
            return;
      }
```

Hem de configurar la cerca automàtica dels jugadors addicionals. També s'ha de configurar l'oient per tal de rebre notificacions dels estats de l'habitació.

També cal reunir els invitats a la llista i buscar els jugadors addicionals. Per fer-ho agafarem els jugadors de la interfície d'usuari de les invitacions.

```
// get the invitee list.
final ArrayList<String> invitees =
      data.getStringArrayListExtra(Games.EXTRA_PLAYER_IDS);

// get auto-match criteria
        Bundle autoMatchCriteria = null;
        int minAutoMatchPlayers =
            data.getIntExtra(Multiplayer.EXTRA_MIN_AUTOMATCH_PLAYERS, 0);
        int maxAutoMatchPlayers =
            data.getIntExtra(Multiplayer.EXTRA_MAX_AUTOMATCH_PLAYERS, 0);

        if (minAutoMatchPlayers > 0) {
            autoMatchCriteria = RoomConfig.createAutoMatchCriteria(
                    minAutoMatchPlayers, maxAutoMatchPlayers, 0);
        } else {
            autoMatchCriteria = null;
        }
```

```
        // create the room and specify a variant if appropriate
        RoomConfig.Builder roomConfigBuilder = makeBasicRoomConfigBuilder();
        roomConfigBuilder.addPlayersToInvite(invitees);
        if (autoMatchCriteria != null) {
            roomConfigBuilder.setAutoMatchCriteria(autoMatchCriteria);
        }
        RoomConfig roomConfig = roomConfigBuilder.build();
        Games.RealTimeMultiplayer.create(mGoogleApiClient, roomConfig);

        // prevent screen from sleeping during handshake
        getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
    }
}

// create a RoomConfigBuilder that's appropriate for your implementation
private RoomConfig.Builder makeBasicRoomConfigBuilder() {
    return RoomConfig.builder(this)
        .setMessageReceivedListener(this)
        .setRoomStatusUpdateListener(this);
}
```

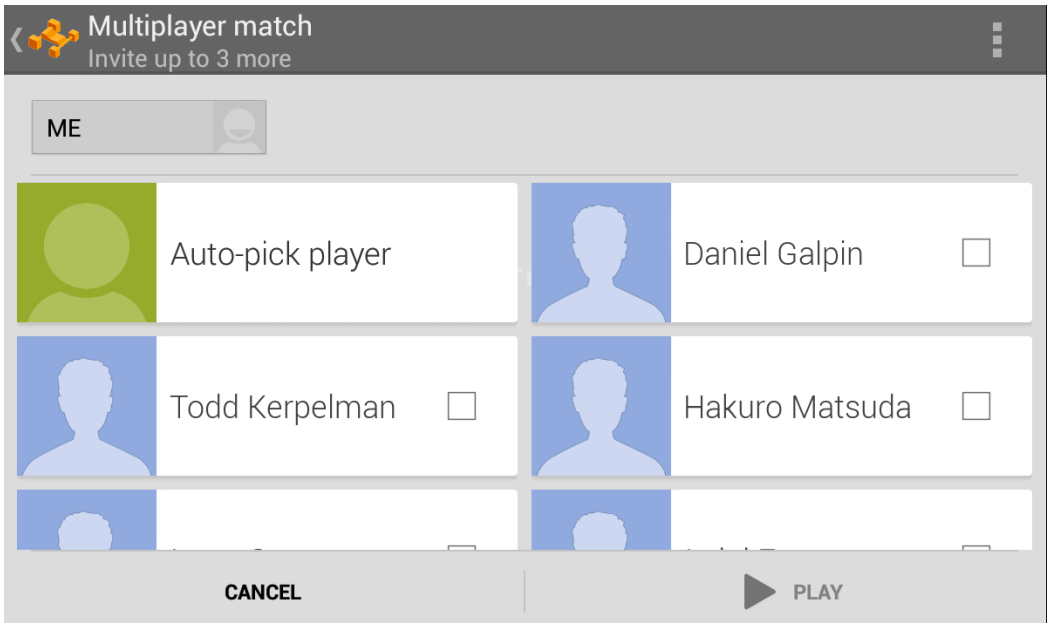


FIGURA 16: EXEMPLE INTERFÍCIE D'USUARI INVITACIONS GOOGLE. RECUPERAT DE HTTPS://DEVELOPERS.GOOGLE.COM/GAMES/SERVICES/ANDROID/REALTIMEMULTIPLAYER

Però, com funciona la cerca de jugadors automàtica?

Diguem que sóc el "jugador 1", llavors faig una partida automàtica, en aquest moment em quedo en espera. He creat una sol·licitud de partida. En aquest moment un "jugador 2" fa el mateix que jo, busca una partida i en el moment que fa la sol·licitud, el servidor ens emparella i llavors s'inicia el joc.

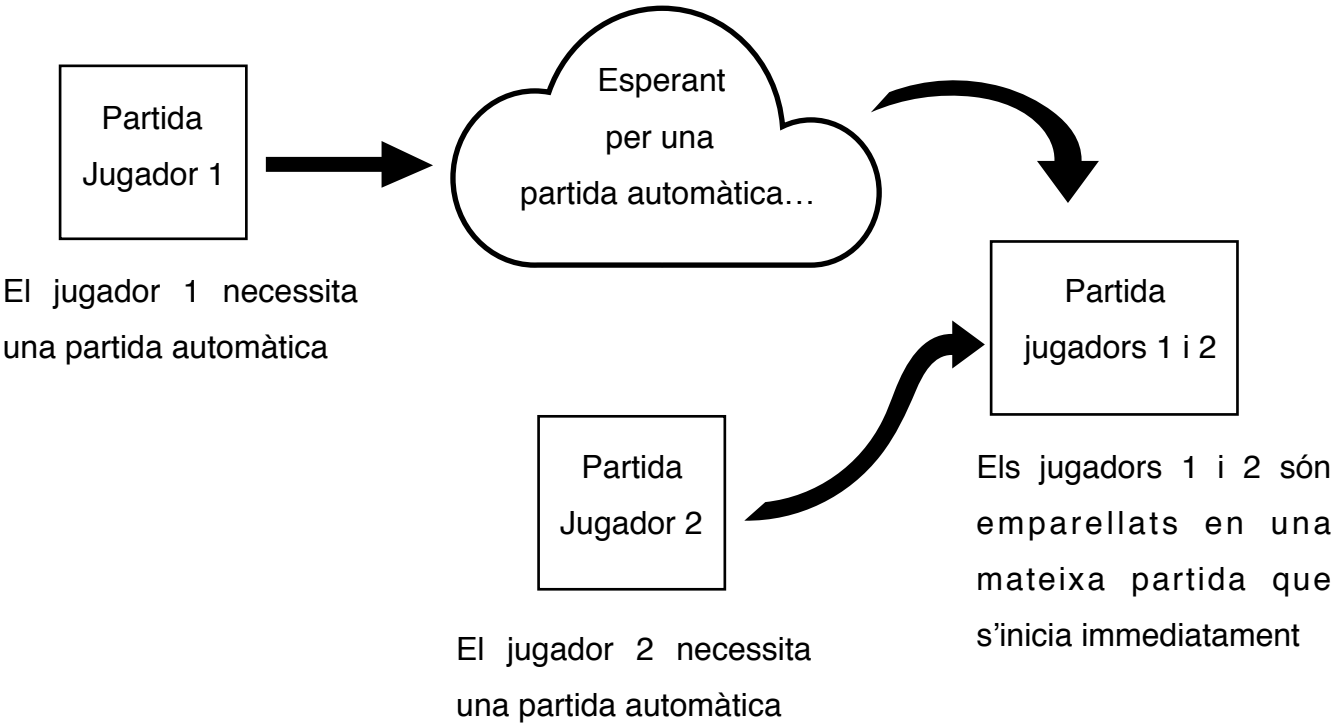


FIGURA 17: EXPLICACIÓ PARTIDA AUTOMÀTICA

Altres opcions

Utilitzar els serveis de Google Play per fer jocs en temps real ens facilita molt les coses, això si, després de superar la corba d'aprenentatge.

Tenim un gran nombre de mètodes que podem sobreescriure per tal de tenir un control més extens de l'aplicació.

Podem manejar els errors de creació d'una habitació utilitzant les crides "callback" de la interfície "RoomUpdateListener" :

- `onRoomCreated(int statusCode,Room room)`
- `onJoinedRoom(int statusCode, Room room)`
- `onRoomConnected(int statusCode, Room room)`

Podem controlar l'estat de connexió dels participants amb alguns mètodes "callback" de l'interfície "RoomStatusUpdateListener" com per exemple:

- `onPeersConnected(Room room, List<String> peers)`
- `onPeersDisconnected(Room room, List<String> peers)`
- `onPeerLeft(Room room, List<String> peers)`
- `onPeersDeclined(Room room, List<String> peers)`

És molt recomanable afegir una interfície de “sala d’espera” per a que els usuaris puguin veure l’estat actual de l’habitació. És molt útil utilitzar-la en crear una sala o quan ens hi unim. Per veure la “sala d’espera” només haurem de cridar l’intent següent:

```
// get waiting room intent
    Intent i = Games.RealTimeMultiplayer.getWaitingRoomIntent(mGoogleApiClient,
room, Integer.MAX_VALUE);
    startActivityForResult(i, RC_WAITING_ROOM);
```

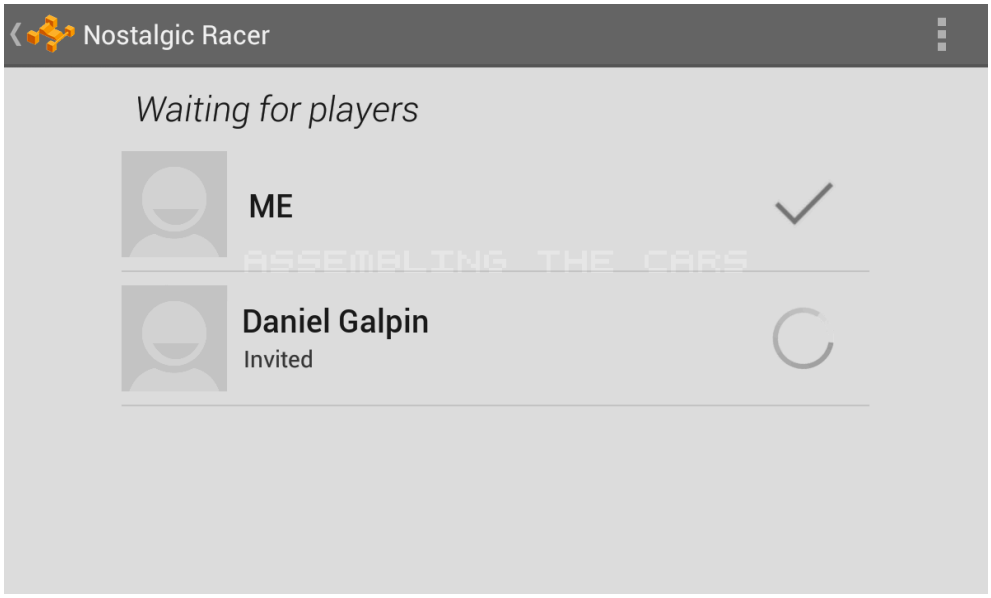


FIGURA 18: SALA D’ESPERA UI. RECUPERAT DE HTTPS://DEVELOPERS.GOOGLE.COM/GAMES/SERVICES/ANDROID/REALTIMEMULTIPLAYER

Haurem de controlar la resposta des d’un oient:

```
@Override
public void onActivityResult(int request, int response, Intent intent) {
    if (request == RC_WAITING_ROOM) {
        if (response == Activity.RESULT_OK) {
            // (start game)
        }
        else if (response == Activity.RESULT_CANCELED) {
            // in this example, we take the simple approach and just leave the
room:
            Games.RealTimeMultiplayer.leave(mGoogleApiClient, null, mRoomId);

getWindow().clearFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
        }
        else if (response == GamesActivityResultCodes.RESULT_LEFT_ROOM) {
            // player wants to leave the room.
            Games.RealTimeMultiplayer.leave(mGoogleApiClient, null, mRoomId);

getWindow().clearFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
```

```
        }
    }
}
```

També podem detectar quan un jugador es desconnecta implementant la crida “callback” “onDisconnectedFromRoom(Room room)”.

Un cop l’usuari ha iniciat sessió pot rebre invitacions. Aquestes invitacions es poden rebre en qualsevol moment, també mentre està jugant, i es guarden en una safata d’entrada. Tot això també ho podem configurar.

Una de les parts més importants dels jocs en temps real és l’intercanvi de dades.
Per poder enviar i rebre missatges primer s’ha de configurar l’habitació cridant “setMessageReceivedListener()” i passant-li l’oient d’argument.
L’enviament de missatges es pot fer amb dos mètodes diferents en funció del tipus de missatge que volem enviar, els mètodes son: “sendReliableMessage()" i “sendUnreliableMessage()”.

```
byte[] message = .......;
for (Participant p : mParticipants) {
    if (!p.getParticipantId().equals(mMyId)) {
        Games.RealTimeMultiplayer.sendReliableMessage(mGoogleApiClient, null,
message,
                mRoomId, p.getParticipantId());
    }
}
```

Quan es rep un missatge és notifica mitjançant la crida.
“RealTimeMessageReceivedListener.onRealTimeMessageReceived“

```
@Override
public void onRealTimeMessageReceived(RealTimeMessage rtm) {
    // get real-time message
    byte[] b = rtm.getMessageData();

    // process message
}
```

Sortirem de la sala quan passi alguna de les següents situacions:

- El joc s’ha acabat
- S’ha fet una crida a “onStop()”, llavors es destrueix la “Activity”.
- L’usuari cancel·la el joc a la “sala d’espera”.

Per marxar només cal cridar “leave()”

```
// leave room
Games.RealTimeMultiplayer.leave(mGoogleApiClient, null, mRoomId);

// remove the flag that keeps the screen on
getWindow().clearFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
```

Implementació final

Un cop ja està tot explicat vegem el resultat final que implementa l'algorisme, fent ús de les operacions criptogràfiques i els serveis de Google.

La majoria de mètodes els he explicat en aquest document en el seu apartat corresponent.

Primerament, en iniciar-se el joc tenim el mètode següent:

```
void firstStep() throws Exception {
     if(!firstSended){
     firstSended = true;
     diceValue = diceResult();
     String seed = randomSeed();
     key = SimpleCrypto.getRawKey(seed.getBytes());
     String cifrado = SimpleCrypto.encrypt(key, String.valueOf(diceValue));
     sendDiceValue(cifrado);
        }
    }
```

En aquest mètode es comprova que no s'ha cridat. Ja veurem mes endavant que també es crida si rebem un missatge de l'altre jugador i nosaltres encara no l'hi hem enviat el nostre.

El mètode "diceResult" el tenim explicat en l'apartat "Implementació de l'algorisme", on també hi tenim la classe "SimpleCrypto".

Vegem els mètodes "randomSeed()" i "sendDiceValue(cifrado)":

```
private String randomSeed(){
       String result = "";
       String upper = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
       String lower = upper.toLowerCase();
       String digits = "0123456789";
       String alphanum = upper + lower + digits; //all possible values
       int size = 32; //size of string
       Random r = new Random();
       while(result.length() < size){
          int i = r.nextInt(alphanum.length() + 1);
          result = result + (String.valueOf(alphanum.charAt(i)));
       }
       return result;
    }
```

Aquest l'hem utilitzat per crear una llavor formada per un conjunts de caràcters aleatoris.

```
 void sendDiceValue(String cifrado){
      // First byte in message indicates that its the initial step
```

```
      mMsgBuf[0] = (byte) ('1');

      // Second byte and  following are the cypher value of dice
      int counter = 1;
      for(char c : cifrado.toCharArray()){
          mMsgBuf[counter] = (byte) c;
          counter++;
      }
      for (Participant p : mParticipants) {
          if (p.getParticipantId().equals(mMyId))
              continue;
          if (p.getStatus() != Participant.STATUS_JOINED)
              continue;
          else {
            // it's an interim notification, so we can use unreliable
           Games.RealTimeMultiplayer.sendUnreliableMessage(mGoogleApiClient,
                  mMsgBuf, mRoomId, p.getParticipantId());
            }
        }

    }
```

Per enviar el valor el que hem fet és utilitzar l'opció que ens dóna Google amb els seus serveis per a jocs en temps real. Enviem una cadena de bytes on utilitzem el primer byte per indicar en quin pas ens trobem de l'algorisme (el pas 1, en aquest cas). A partir del segon byte hi tenim la clau xifrada.

Per rebre aquest missatge i els que enviarem més endavant necessitem implementar un oient que estarà esperant rebre un missatge. També ens el proporciona Google i el mostrem a continuació:

```
@Override
public void onRealTimeMessageReceived(RealTimeMessage rtm) {
      byte[] buf = rtm.getMessageData();
      if(buf[0] == '1'){
          try {
              if (!firstSended){
                  firstStep();
              }
          }catch (Exception e){
              e.printStackTrace();
          }
          receiveDice(buf);
          sendKey();
      }else if (buf[0] == '2') {
          receiveKey(buf);
      }
}
```

El sobreescrivim per fer les comprovacions per saber en quin pas estem de l'algorisme (1 o 2).

Si estem en el primer pas, hem rebut el valor de la tirada xifrat. Ara caldrà, si encara no l'hem enviat nosaltres, enviar el nostre valor. A més haurem de tractar el valor rebut amb el mètode "receiveDice(buf)", que el tenim detallat a l'apartat "Implementació de l'algorisme". Després, amb el mètode "sendKey()", enviem la clau que hem utilitzat per xifrat.

Si estem en el segon pas, hem rebut la clau que l'altre jugador ha utilitzat per xifrar. El que farem és tractar la clau, i acabar l'algorisme amb el mètode "receiveKey(buf)", el qual tenim detallat a l'apartat "Implementació de l'algorisme".

Vegem la funció:

```
void sendKey(){
     // First byte in message indicates that its the second step
     mMsgBuf[0] = (byte) ('2');
     // Second byte and followings are the key
     int counter = 1;
     for(byte b : key){
         mMsgBuf[counter] = b;
         counter++;
     }
    for (Participant p : mParticipants) {
        if (p.getParticipantId().equals(mMyId))
            continue;
        if (p.getStatus() != Participant.STATUS_JOINED)
            continue;
        else {
           // it's an interim notification, so we can use unreliable
           Games.RealTimeMultiplayer.sendUnreliableMessage(mGoogleApiClient,
            mMsgBuf, mRoomId, p.getParticipantId());
       }
    }
}
```

Aquí podem veure com enviem la clau. Com abans, el primer byte ens indica que estem en el segon pas i a continuació hi posem la clau. Finalment ho enviem mitjançant el mètode "sendUnreliableMessage" que ens proporciona Google.

Si volguéssim generalitzar aquest algorisme a més de 2 jugadors hauríem de reservar un byte més al buffer per indicar quin jugador envia el missatge i, a més, guardar els diferents resultats i claus en, per exemple, una cadena de claus i un altra de valors. També caldria modificar el mètode de càlcul final per tractar més valors.

Conclusions

En aquest treball hem implementat un algorisme de llançament de dau segur en jocs distribuïts i després l'hem utilitzat en una aplicació Android on hem après a utilitzar els serveis de Google Play Games.
Parlant de l'algorisme trobo que és una manera molt fàcil de garantir l'aleatorietat de cada tirada i que hem aconseguit el que preteníem.
En quant a l'aplicació, més enllà de la feina que m'ha portat, ja que sense dubte ha estat la part on he posat més hores, penso que he après moltíssim sobre programació Android. És cert que a classe havíem fet molt temari, però tenia pendent el tema dels serveis de Google, i ara que ja sé com funciona, probablement quan tingui temps lliure em posaré a desenvolupar algun joc pel meu compte.
En resum, aquest treball m'ha donat feina però m'ha aportat més del què pensava i estic força content de haver-lo acabat!

Com a futures línies de treball, en principi no hi hauria més ja que l'aplicació està acabada. Tot i així trobo que podria aprofitar tot el que he après sobre els serveis de Google per fer una altra aplicació que tractés d'un joc amb més tirada que aquest, i així aconseguir un gran nombre d'usuaris. Aquest joc no compleix aquesta finalitat ja que va lligat amb la primera part i depeníem molt del temps. Seguint aquest punt en un futur es podria aprofitar l'algorisme per qualsevol joc distribuït en que sigui necessari el llançament d'un dau, com per exemple un parxís. A més, aquest algorisme es pot extrapol·lar a molts altres jocs d'atzar canviat el dau per allò que determini el factor atzarós d'aquest joc, com podria ser una moneda o una ruleta.

Webgrafia i bibliografia

1. Documentació sobre el mètode de xifrat AES: https://ca.wikipedia.org/wiki/Advanced_Encryption_Standard#Desenvolupamentt

2. Guía ràpida d'inici als serveis de Google Games: https://developers.google.com/games/services/android/quickstart

3. Suport de Google per als jocs Android multi-jugador en temps real https://developers.google.com/games/services/android/realtimeMultiplayer

4. Configuració dels Serveis de Google: https://developers.google.com/games/services/console/configuring#enabling_multiplayer_support

5. Control de versions i exemples similars: https://github.com/

6. Icones per l'aplicació: https://material.io/icons/

7. Documentació de Java: https://docs.oracle.com/javase/7/docs/api/

8. Documentació Android: https://developer.android.com/index.html

9. Francesc Sebé. Apunts de l'assignatura seguretat d'aplicacions i comunicacions. 4r curs d'enginyeria informàtica a la UdL.